Improving Single-Thread Performance through Core Overclocking

Brian Greskamp and Josep Torrellas

http://iacoma.cs.uiuc.edu/
Motivation

- Frequency growth is slowing
- CMP trend
  - Moderate IPC per core
  - Plenty of cores to spare
- How to improve single-thread performance?
Paceline Concept

Many-Core CMP
Paceline Concept

Leader

Program Thread

Checker

Paceline: Brian Greskamp
Paceline Concept

- Overclock
- Hints
- Safe Clock
- Leader
- Checker

Paceline: Brian Greskamp
Paceline Concept

Overclock

Leader

Safe Clock

Checker

Hints

Paceline: Brian Greskamp
Paceline Concept

Overclock

Leader

= ?

Hints

Checker

Safe Clock

Paceline: Brian Greskamp
Paceline Concept

Overclock

Leader

= ?

Hints

Checker

Safe Clock

Paceline: Brian Greskamp
Paceline Concept

Overclock → Swap → Safe Clock

Leader

Hints

Checker

Paceline: Brian Greskamp
Paceline Concept

Safe Clock  Periodically swap  Overclock

Checker ≡ Hints

Leader

Paceline: Brian Greskamp
Paceline Concept

Periodically swap

Overclock → Safe Clock

Leader

Hints

Checker

Hints

Paceline: Brian Greskamp
Paceline Concept

Safe Clock

Thread 0

Hint

Thread 1

Safe Clock

Paceline: Brian Greskamp
Contributions

• Paceline: A new approach to improving single-thread performance through replication
• Two detailed microarchitecture implementations
  • High performance
  • Minimal power and thermal impact
  • Optional additional fault tolerance
Potential for Overclocking

- Processor speed grades are determined by:
  - The slowest core on the CMP
  - ... under worst-case environmental conditions
  - ... at the point of first failure
Potential for Overclocking

- ... the slowest core
- Process variation causes core-to-core frequency variation
- Speed grading places all chips into pre-determined bins

<table>
<thead>
<tr>
<th>CMP with Variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.62</td>
</tr>
<tr>
<td>3.71</td>
</tr>
<tr>
<td>4.05</td>
</tr>
</tbody>
</table>

$\frac{f_{\text{max}}}{f_{\text{rated}}} = \frac{3.62}{3.50}$
Potential for Overclocking

• ... under worst case conditions
• Circuits slow with:
  • Age (NBTI, HCI)
  • Increasing temperature (activity, environment)
  • Supply voltage droop ($L \frac{dI}{dt}$, $IR$)

Paceline: Brian Greskamp
Overclockability

- ... at the point of first failure
- Redundancy can detect and correct occasional timing faults
Paceline: Achieving Speedup

- Note: Cores don’t operate in lockstep; leader runs ahead
- Hints from leader improve checker performance (IPC)
  - Branch outcomes
  - Prefetches
Paceline:
Ensuring Correctness
Paceline: Ensuring Correctness

- Periodically take register checkpoints
Paceline: Ensuring Correctness

- Periodically take register checkpoints
- Add VQ to compare leader and checker states

Diagram:
- Leader
  - Req Ckpt
  - Hash
  - L1
  - BQ
- Checker
  - Req Ckpt
  - Hash
  - L1
  - VQ
  - Coherent L2

Paceline: Brian Greskamp
Paceline: Ensuring Correctness

- Periodically take register checkpoints
- Add VQ to compare leader and checker states
- No I/O or write is performed at L2 until it has been checked
Paceline: Thermal Management

- Leader consumes more power than a base core.
- Often, checker consumes less power than base core.
Paceline: Thermal Management

- Periodically swapping cores averages temperature
- Improves reliability, overclocking potential
- Minimizes leakage power
### Types of Faults

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Repeats during Re-Executions?</th>
<th>Leader Impact</th>
<th>Checker Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing Error due to Overclock</td>
<td>Likely</td>
<td>Register, L1 State</td>
<td>None</td>
</tr>
</tbody>
</table>

*Paceline: Brian Greskamp*
Input Incoherence

- Both cores’ L1s send normal read misses to L2
- Other core’s write can cause leader and checker to read different values ("incoherence") [Slipstream02][Reunion06]
# Types of Faults

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Repeats during Re-Executions?</th>
<th>Leader Impact</th>
<th>Checker Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing Error due to Overclock</td>
<td>Likely</td>
<td>Register, L1 State</td>
<td>None</td>
</tr>
<tr>
<td>Input Incoherence</td>
<td>Possibly</td>
<td>Register, L1 State</td>
<td>None</td>
</tr>
<tr>
<td>Soft Error</td>
<td>No</td>
<td>Register, L1 State</td>
<td>Register, L1 State</td>
</tr>
</tbody>
</table>
Paceline Architecture

• Two Paceline design points:
  • *Simple* guarantees correct execution if checker has no soft errors
  • *High-reliability* tolerates checker soft errors
  • Both provide improved performance
  • Both tolerate all leader faults
Simple Microarchitecture

Assume Checker is always correct

Leader

Checker

Reg Ckpt

Hash

BQ

VQ

Reg Ckpt

Hash

L1

L1

Paceline: Brian Greskamp
**Simple Microarchitecture**

Assume Checker is always correct

- VQ buffers and compares register value hashes
**Simple Microarchitecture**

Assume Checker is always correct

- VQ buffers and compares register value hashes
- Only checker writes to L2
Simple Microarchitecture

Assume Checker is always correct

- VQ buffers and compares register value hashes
- Only checker writes to L2
- Leader write handling options
  - Drop writebacks
  - Use VQ as victim cache
Simple Microarchitecture: Recovery

Assume Checker is always correct

On error, copy state from Checker
Simple Microarchitecture: Recovery

Assume Checker is always correct

On error, copy state from Checker

- Roll leader forward past fault
  1. Copy checker regfile contents to leader
Simple Microarchitecture: Recovery

Assume Checker is always correct

On error, copy state from Checker

- Roll leader forward past fault
  1. Copy checker regfile contents to leader
  2. Flush leader L1 cache
Simple Microarchitecture: Recovery

Assume Checker is always correct
On error, copy state from Checker

- Roll leader forward past fault
  1. Copy checker regfile contents to leader
  2. Flush leader L1 cache
  3. Clear VQ, BQ
High Reliability Microarchitecture

Assume Checker can experience soft errors
High Reliability Microarchitecture

Assume Checker can experience soft errors

- VQ buffers and compares register hashes
High Reliability Microarchitecture

Assume Checker can experience soft errors

- VQ buffers and compares register hashes
- VQ buffers and compares write values
- L1s are Write-Through
High Reliability Microarchitecture

Assume Checker can experience soft errors

- VQ buffers and compares register hashes
- VQ buffers and compares write values
- L1s are Write-Through
- VQ releases writes to L2 after successful comparison of checkpoint interval
High Reliability Microarchitecture: Recovery

Assume Checker can experience soft errors

On error, roll back both cores
High Reliability Microarchitecture: Recovery

Assume Checker can experience soft errors

On error, roll back both cores

1. Restore register checkpoint in both cores
High Reliability Microarchitecture: Recovery

Assume Checker can experience soft errors

On error, roll back both cores

1. Restore register checkpoint in both cores
2. Flush both caches
High Reliability Microarchitecture: Recovery

Assume Checker can experience soft errors

On error, roll back both cores

1. Restore register checkpoint in both cores
2. Flush both caches
3. Invalidate all entries in VQ from the failing checkpoint
High Reliability

Microarchitecture: Recovery

Assume Checker can experience soft errors

→ On error, roll back both cores

1. Restore register checkpoint in both cores
2. Flush both caches
3. Invalidate all entries in VQ from the failing checkpoint
4. Restart both cores with safe clock frequency
Special Cases

• Interrupt: Deliver at next checkpoint
• Read-Modify-Write: Leader sends read to prefetch the line, but only checker can perform write
• Repeated incoherence: Techniques to avoid livelock in paper
Evaluation
**Experiment Setup**

- SESC simulator with WATTCH power modeling

<table>
<thead>
<tr>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>16 OoO cores, 32nm, 5 GHz</td>
</tr>
<tr>
<td>Core width</td>
<td>6 fetch, 4 issue, 4 retire</td>
</tr>
<tr>
<td>ROB size</td>
<td>152</td>
</tr>
<tr>
<td>Scheduler size</td>
<td>40 fp, 80 int</td>
</tr>
<tr>
<td>LSQ size</td>
<td>54 LD, 46 ST</td>
</tr>
<tr>
<td>Branch pred</td>
<td>80Kb local/global tournament, unbounded RAS</td>
</tr>
<tr>
<td>L1 I cache</td>
<td>16KB, 2 cyc, 2 port, 2 way</td>
</tr>
<tr>
<td>L1 D cache</td>
<td>16KB WT, 2 cyc, 2 port, 4 way</td>
</tr>
<tr>
<td>L2 cache</td>
<td>2MB WB, 10 cyc, 1 port, 8 way, shared by two cores, has stride prefetcher</td>
</tr>
<tr>
<td>Cache line size</td>
<td>64 bytes</td>
</tr>
<tr>
<td>Memory</td>
<td>400 cyc round trip, 10GB/s max</td>
</tr>
</tbody>
</table>
Paceline Speedup

Overclocking Factor: □ 10% □ 20% □ 30% □ 40%

Large speedups for SPECint: 1.21 at 30% overclocking
Paceline Power

Two base cores

Paceline power \approx \text{Power of two base cores}

Paceline: Brian Greskamp
Conclusions

Paceline: A new approach to improving single-thread performance through replication

- Large performance gains
- Minimal power and thermal impact
- (Optionally) Improved fault tolerance
Improving Single-Thread Performance through Core Overclocking

Brian Greskamp and Josep Torrellas

http://iacoma.cs.uiuc.edu/
Performance Analysis

Obtain from any cycle-accurate simulator

$L_i$: Instantaneous Leader speedup at interval $i$
assumptions: perfect (infinite-speed) checker

$C_i$: Instantaneous Checker speedup at interval $i$
assumptions: perfect (infinite-speed) leader

Generate Paceline speedup estimate

$P_i$: Instantaneous Overall speedup at interval $i$

$S_j$: Average Overall speedup at interval $j$

$$P_i = \min(L_i, C_i).$$

$$S_j = \frac{j}{\sum_{i=1}^{j} \frac{1}{P_i}}$$

Paceline: Brian Greskamp
Performance Analysis: Examples

\[ L_i \quad C_i \quad \min(L_i, C_i) \quad S_j \]

Speedup

Dynamic Inst #

Dynamic Inst #

Dynamic Inst #

Dynamic Inst #

Dynamic Inst #

Dynamic Inst #

Dynamic Inst #

Dynamic Inst #

Dynamic Inst #

Dynamic Inst #

Dynamic Inst #

Dynamic Inst #
Performance Simulation: Overclocking Factor

![Graph showing speedup vs. leader overdrive for SPECint, SPECfp, SPECfp-noBQ, and SPECInt-noBQ benchmarks. The graph indicates that increasing overdrive leads to increased speedup, with SPECInt-noBQ showing the greatest speedup gain. A yellow box highlights that the Branch Queue is critical to speedup.]
Power Analysis

Obtain from any energy-enabled simulator

\( E_l \): Total leader energy (overclocked)
\( E_c \): Total checker energy (perfect cache, perfect bpred)
\( S \): Overall Paceline speedup
\( T_b \): Execution time on baseline core

Generate Paceline core power estimate

\[ P = \frac{S (E_l + E_c)}{T_b} \]

• Does not include energy for Paceline structures
**Performance Sensitivity**

![Graph showing performance sensitivity](image)

Recovery penalty is not critical to performance

*Paceline: Brian Greskamp* 57